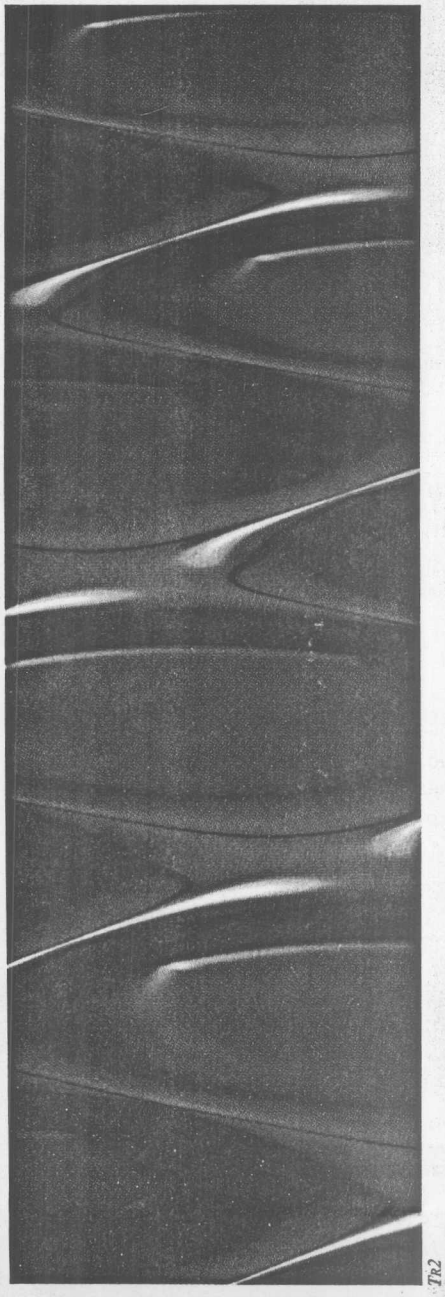
by CLAY DEARMAN

The Practical FFT

Fast Fourier Transforms may be a headache to understand, but they may not be so difficult to implement. This article describes how you can use an FFT in a microcontroller application.



Volumes have been published about the fundamental workings of the Fast Fourier Transform (FFT), but rarely do these publications address the real usefulness of the algorithms that they explain. Programmers do not have to be able to derive FFTs for themselves before they can make use of them. If you grasp the fact that the FFT is just an efficient algorithm for computing a Discrete Fourier Transform (DFT) and that the DFT transforms a series of time domain samples into a series of frequency domain samples, you are ready to put the FFT to good use.

FFTs have been written for just about every processor and platform, and many of these are available on bulletin boards around the country. Once you understand what the FFT will and will not do and you know the format of the input and output data for the transform you have chosen, you are ready to make use of this new tool. Before we jump into an actual application, let's take a look at what happens when we transform a signal into the frequency domain.

TRANSFORM BASICS

The DFT operates on finite sets of data taken at discrete points in time. This differs from the waveforms you want to transform in that they are continuous in time. These waveforms must be sampled at discrete time intervals before the DFT or FFT can be applied. Most likely, your waveform is not infinite in time, but the chances are good that you will not be able to sample your waveform over its entire duration. Instead, you will have to limit your number of samples, thus presenting only a portion of the actual waveform to the transforming algorithm. The analog-to-digital conversion process subjects the data presented to the transform to two basic phenomena: windowing and sampling.

Let's take a cosine wave that is infinite in time. Obviously, we cannot fit the entire waveform into the computer, so we'll take 32 samples of this waveform at 0.25-second intervals. The result is that we have multiplied the infinite waveform by a window eight seconds long and by a sampling train. This sampling train can be represented by a series of 32 impulse functions with an amplitude of one and spaced 0.25 seconds in time. Figure 1 graphically shows the results of windowing and sampling our infinite waveform. The windowed and sampled waveform can be transformed to the frequency domain by applying the Fourier integral over the window interval to the product of the original waveform and the sampling train. The result of windowing and sampling the input data is that the Fourier integral now becomes the DFT.

Windowing and sampling also have effects on the output of the transform. Windowing can cause leakage errors in the transform's output. Figure 2 represents a pure cosine wave along with its frequency domain magnitude. Exactly 12 cycles of the cosine wave were captured in the time window.

FIGURE 1
The waveform sampling process.

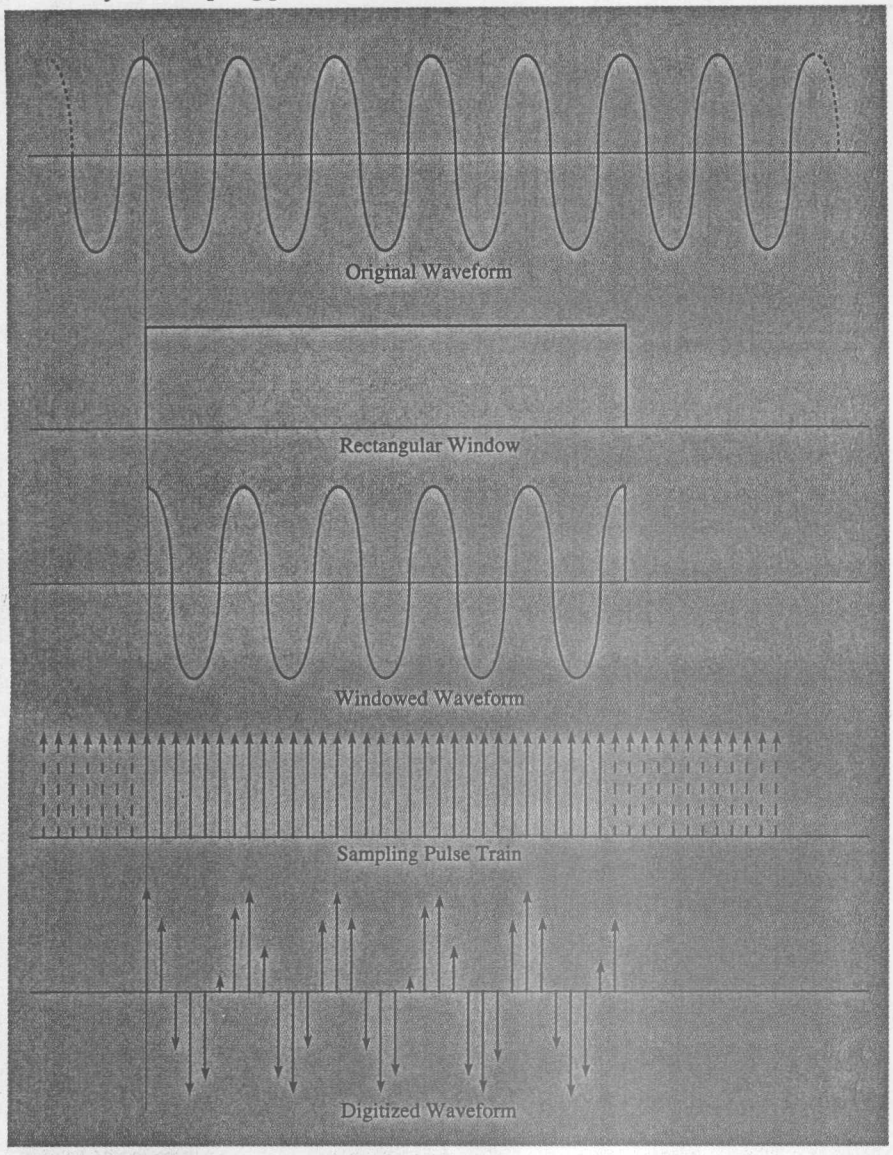
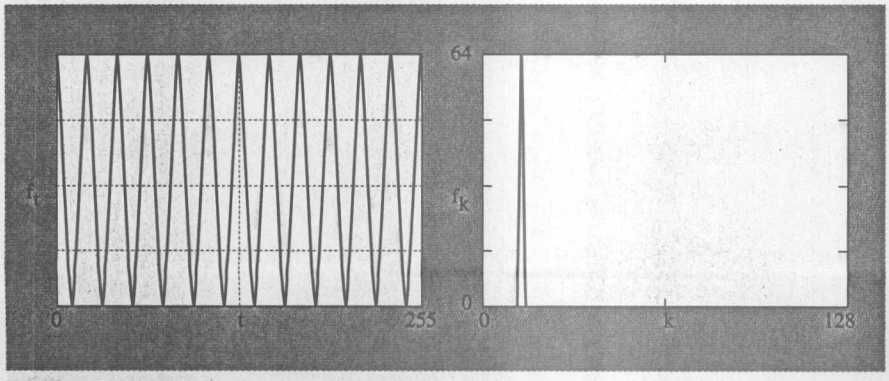
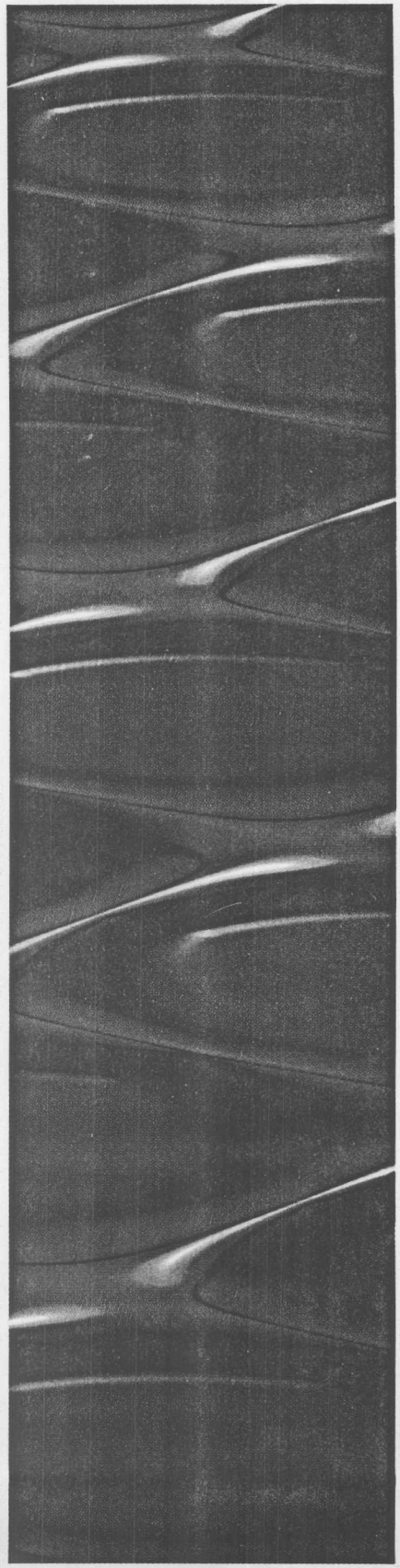


FIGURE 2
Time sample of exactly 12 cycles and the corresponding FFT output.





Each of the frequency domain sample points falls at a zero, with the exception of the fundamental frequency.

Figure 3 represents a cosine wave and its frequency domain magnitude, but this time, 12.5 cycles were captured. Simply changing the number of cycles of the captured input waveform caused a marked difference in frequency domain magnitudes between Figures 2 and 3. This is an example of leakage. A cosine wave, sampled so an integer number of cycles is captured in the window, will transform into the frequency domain as shown in Figure 4a. But what about the window we applied? Let's transform it to the frequency domain and then multiply it with our cosine wave. Figure 4b shows the window in the time and frequency domain. Figure 4c shows the windowed cosine wave. Sampling is introduced in Figure 4d, and Figure 4e illustrates the results. Each of the frequency domain sample points falls at a zero, with the exception of the fundamental frequency.

This happened because the cosine wave is harmonically related to the window length. In English, there were an integer number of cycles in the window. When a noninteger number of cycles is in the window, the frequency samples do not line up very well with the continuous spectra, as shown in Figure 5. Also, the remaining frequency domain samples fall on nonzero portions of the side lobes. This is leakage error. It takes power from compo-

FIGURE 3
Time sample of 12.5 cycles and the corresponding FFT output.

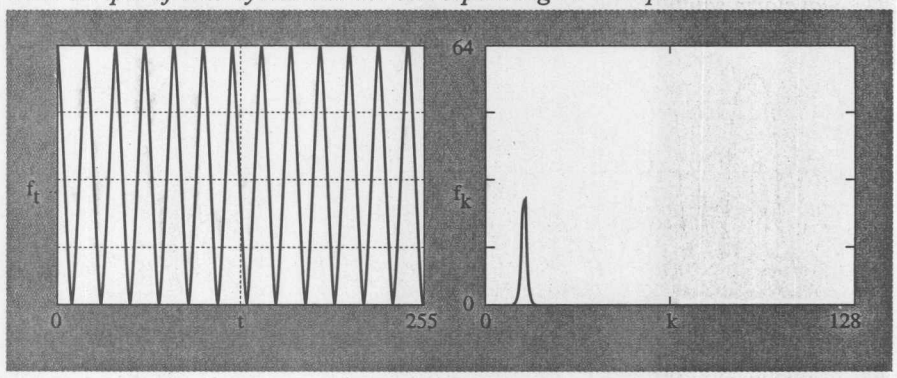


FIGURE 4
Integer number of cycles.

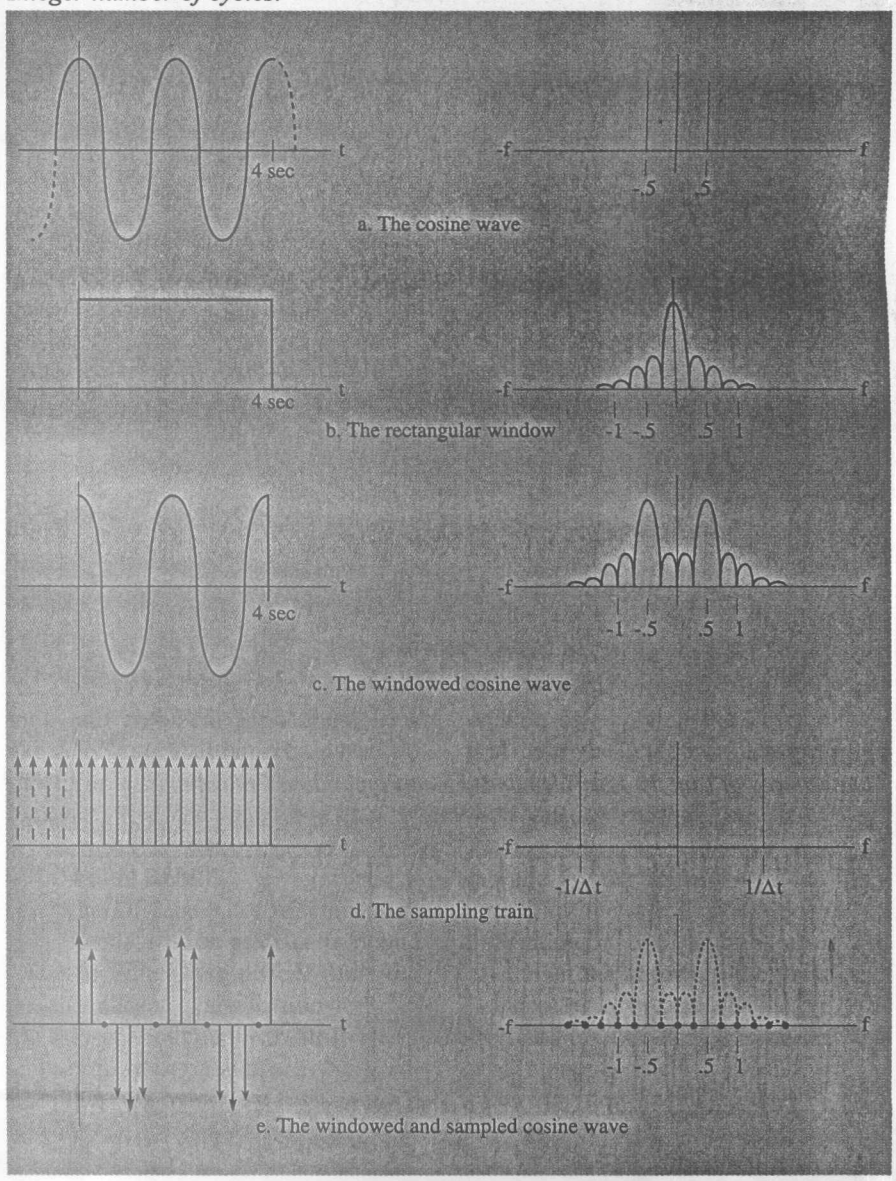
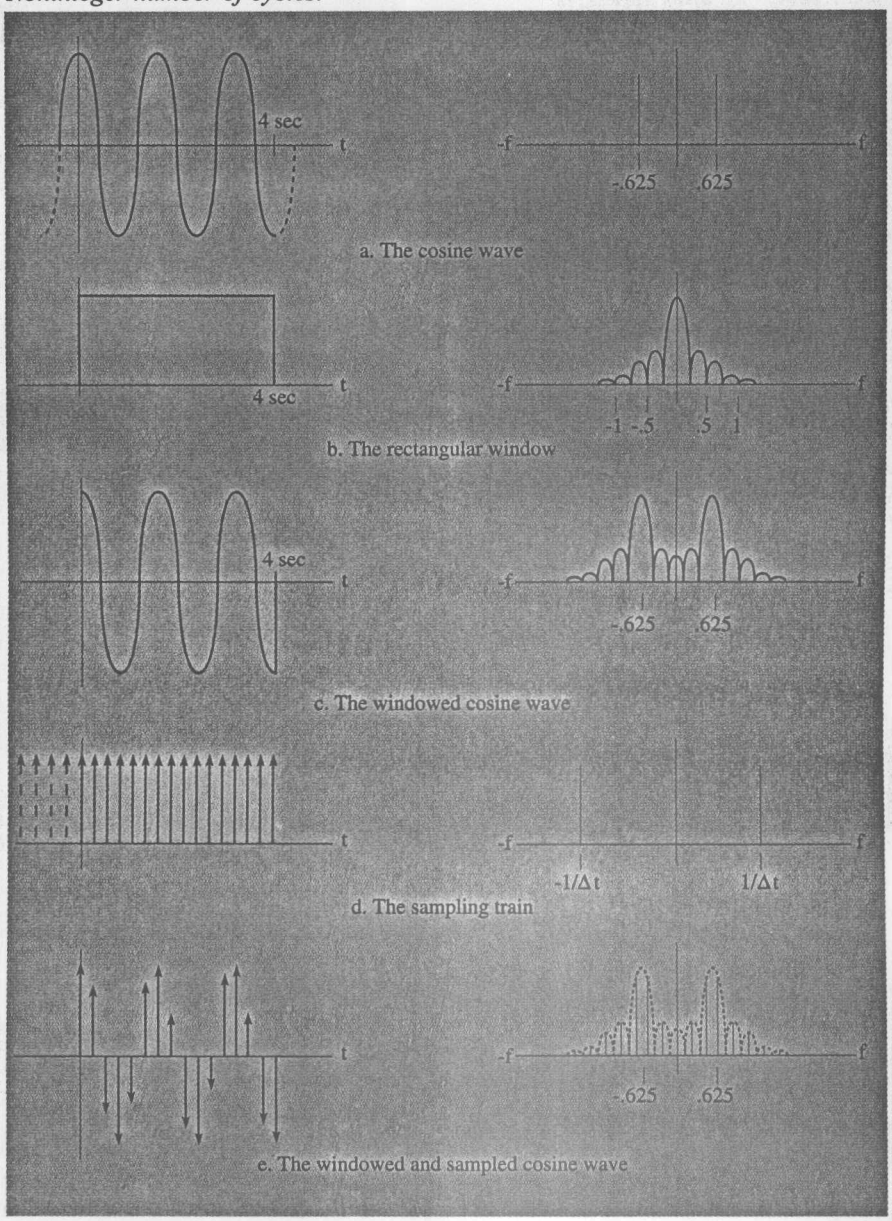


FIGURE 5
Noninteger number of cycles.



nents in the continuous waveform and gives it to nonexistant components. The shape of the leakage error is determined by the shape of the window. Many different windows have been developed to reduce the effect of leakage. Obviously, the rectangular window is the easiest to apply, but others deserve some attention. The Hanning window described in Figure 6 is a simple and popular window. If we multiply the time domain waveform by the Hanning window, the discontinuity at the end of the waveform is removed because each end is pinched to zero, and the leakage error shown in Figure 7 is reduced.

IMPLEMENTATION ISSUES

Now that we understand what the FFT will do for us, let's take a look at some ways to implement it. The processor you use to perform the FFT does not have to be a genius. The FFT's usefulness in embedded systems is not limited to the

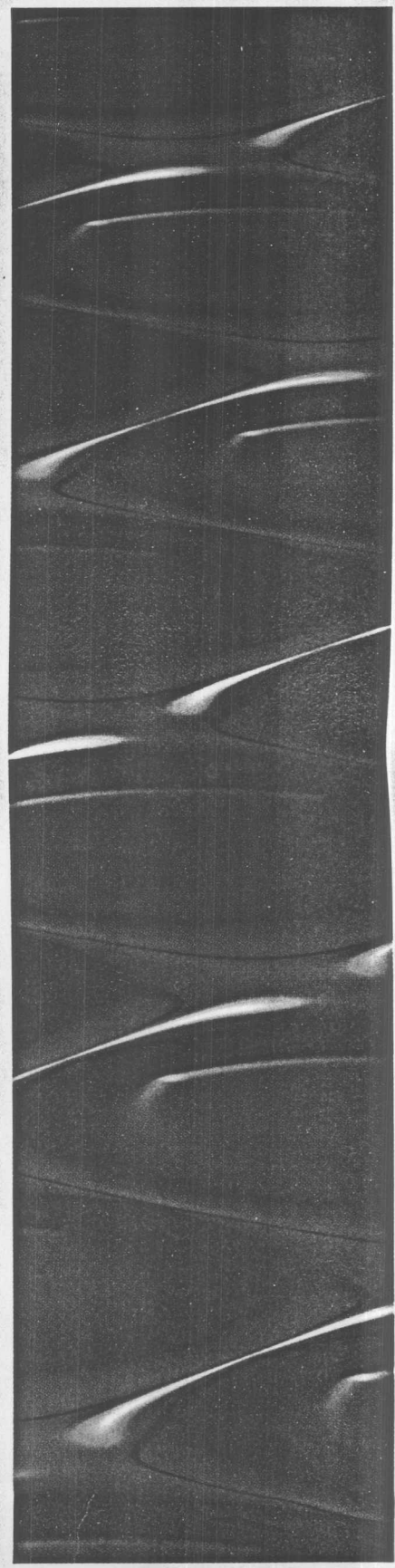


FIGURE 6
Time domain view of the Hanning window.

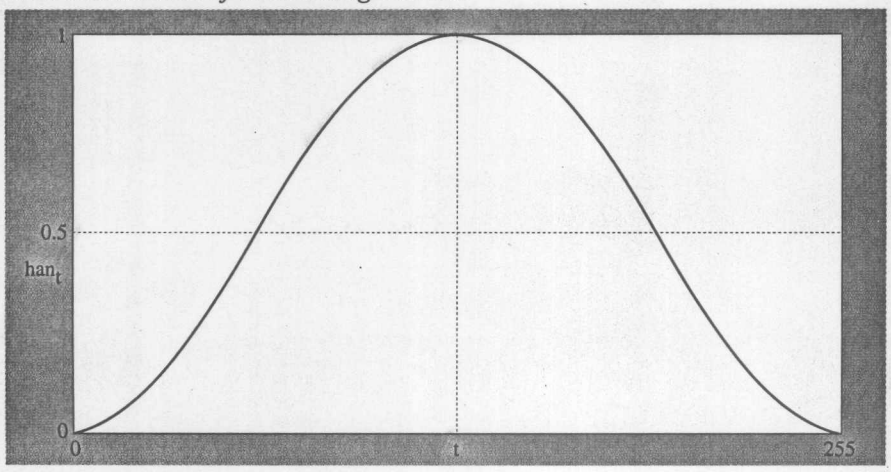
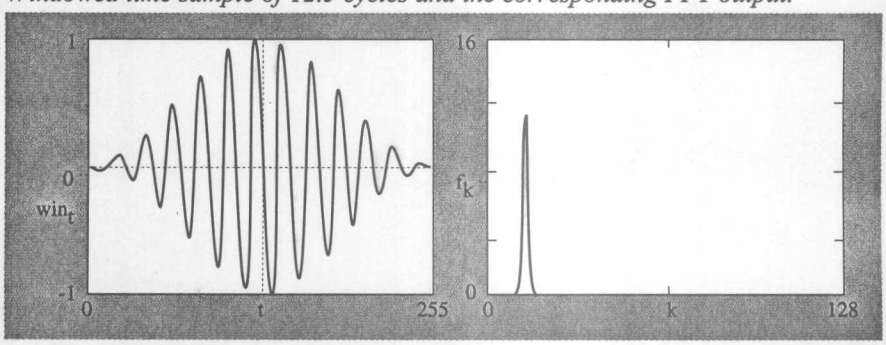


FIGURE 7
Windowed time sample of 12.5 cycles and the corresponding FFT output.



so-called digital signal processors (DSPs), nor does it have to be implemented using floating-point math. Even simple 8-bit processors can quickly perform the math needed to implement a fixed-point limited array size FFT. This realization allows the FFT to find a home in projects not typically considered to be DSP-based.

There are an unlimited number of applications for the FFT in embedded systems. FFTs can be used to monitor bearing or drivetrain conditions in mechanical systems or for the detection of analog signals in mixed-signal electrical systems. These applications can provide increased product features or performance by taking advantage of hardware already in place.

Several criteria must be considered before trying to implement an FFT-based solution, however. The sampling rate of the data acquisition software must be fast enough to handle the frequency range you need to inspect. A minimum of two samples must be taken during every period of the highest frequency you intend to capture (the Nyquist rate), and if higher frequencies than this exist on the input, a low-pass filter must be installed on the input to prevent the higher frequencies from aliasing down into the frequency range being examined. This means that the time between samples must be no more than:

SR = (1/Fmax)/2

If large dynamic ranges are important to the result, higher resolution analog-to-digital conversion will be required as well as higher resolution math. Frequency resolution will depend on the sampling rate and on the number of points (samples) the FFT is designed to handle. The more points in the FFT, the greater the frequency resolution and the more time-consuming the calculations. The key is to decide what is good enough. As mentioned earlier, leakage error can be a problem, but it may be possible to define your sampling rate so an integer number of cycles of the incoming waveform is captured. This can be done only if you know all of the frequencies that could possibly be present in the incoming waveform and you can set your window size (sampling rate X number of samples) to provide a frequency domain sample that is an even factor of each frequency expected. If you do not know the frequencies present or the only common factor is one, using a window shape other than rectangular may help out. If you are simply looking for the presence of a certain frequency in the incoming signal, leakage may not make any difference. Again, it all depends on the application.

If you are looking for the presence of a certain frequency, leakage may not make any difference.

A PRACTICAL IMPLEMENTATION

A recent project I worked on included, among other things, the need to detect different call progress tones emitted from the earpiece of a telephone. The tones that had to be detected—a dialtone, a busy signal, and a ringing line—were supposed to be decoded by a hardware device, but it proved to be too slow and

unreliable. The device required over two seconds to make a decision, an absolute eternity when trying to place a phone call. After some head scratching, I decided to give the FFT a try. In this case, each tone that needed detection consisted of two separate frequencies. All I had to do was acquire the data, perform an FFT on it, and check the amplitude of the appropriate locations (bins) in the result for a value of at least what I considered to be a valid amplitude.

Sounds simple? Well, it was. Since my telephone was controlled by my favorite MC68HC811E2, I dialed up the Motorola bulletin board and downloaded an FFT written for the 'HC11. It was written for 256 data points and used fixed-point math. Since I was not concerned with absolute amplitude accuracy, I did not need to window my data. I did, however, have to take the unsigned data provided by the 'HC11's 8-bit analog-to-digital converter (ADC) and convert it to signed data before presenting it to the FFT. The highest frequency I needed to convert was 625 Hz, so my sampling rate could be no slower than 800 μs. The following equation relates the sampling rate to the frequency resolution of each output bin of the FFT. Remember that a 256-point FFT yields 128 bins in the positive frequency realm.

1/(SR*num samples) = freq resolution

All of the frequencies I needed to decode were multiples of five Hz, so I selected a sampling rate of 778 μs, giving a frequency resolution of about five Hz per bin (1/(778 μs * 256). The data acquisition software is shown in Listing 1, ACQ. It must be run with interrupts disabled, because any asynchronous events will skew the timebase. The acquisition routine is very simple in nature, using one of the 'HC11's output compare timers to space each data conversion. Remember, in the 'HC11, the ADC will complete four conversions before setting the conversion complete flag,

LISTING 1
Time domain acquisition.

```
        ORG     $1000           ; Set to start of RAM

DATA    RMB     512             ; Set aside data location

        ORG     $C000           ; Set to start of code

* ACQ - This routine acquires the time domain data. It uses a sampling rate of 778 uS so
* that each bin of the FFT is about 5 Hz.

ACQ
        SEI                     ; Disable interrupts
        LDX     #DATA           ; Get the address of start of data

ACQ3    LDD     TCNT            ; Get the current count
        ADDD    #89             ; Add in 712 uS. This plus the code delays
                                ; and the A/D conversion time add up to
                                ; approximately 778 uS
        STD     TI405           ; Start the timer ticking
        BCLR    TFLG1 %11110111 ; Clear the timer flag

ACQWT1  BRCLR   TFLG1 %00001000 ACQWT1      ; Wait for timer to expire
        LDAB    #0              ; Start A/D
        STAB    ADCTL

ACQWT2  BRCLR   ADCTL %10000000 ACQWT2      ; Wait for conversion complete
        LDAB    ADR1            ; Get the result
        STAB    0,X             ; Save it
        INX
        CPX     #DATA+256
        BLO     ACQ3            ; If not done, try another
        CLI                     ; Reenable interrupts
        RTS                     ; All done
```

LISTING 2
Shift.

```
* SHIFT - This routine takes the unsigned 256 data points located at DATA and shifts it
* about its median. The median is calculated as half the difference between the max and
* the min. Doing this will yield the signed data that the FFT expects.
SHIFT
        LDX     #DATA           ; Get the address of the data
        LDAA    0,X             ; Put the first value in the max register
        LDAB    0,X             ; and in the min register
SHIFT1  INX
        CMPA    0,X             ; Check for new max value
        BHS     SHIFT2
        LDAA    0,X             ; If so then get it
SHIFT2  CMPB    0,X             ; Check for new min value
        BLS     SHIFT3
        LDAB    0,X             ; If so then get it
SHIFT3  CPX     #DATA+255
```

which will take 128 E clock cycles. I was running with an 8-MHz crystal, resulting in 500 ns per clock cycle and 64 μs from the time the ADC is started to the time it completes.

If extreme accuracy is important, you should measure your clock so these calculations will be correct. After the time domain data has been collected, it must be converted to signed data. Listing 2 shows the routine SHIFT that does this conversion. SHIFT finds the maximum and the minimum of the data, then shifts the data so that the median value ((max-min)/2) becomes zero. If the zero offset of the data is known before the data is collected, the shift could be done at the time of acquisition, saving the execution time of doing it after collection. Since I was not pressed for performance, I chose to center my data after collection, allowing me to throw away any DC offset fluctuations. After shifting the data, I invoked the FFT. Again, I did not write this FFT. It is readily available to anyone on Motorola's Freeware Data System's BBS at (512) 891-3733.

In the header of the FFT routine, there is a pseudo "power spectra" computation routine that can be used to furnish a magnitude instead of real and imaginary data points. It calculates the sum of the absolute values instead of the sum of the squares, but that is just fine for what I needed. After performing the FFT and calculating the "power spectra," CHECK (Listing 3) is called to look for the frequencies of interest. Experimentation led me to choose the threshold level of $28. Since I only needed to detect the presence of certain frequencies, I made no effort to calibrate the amplitude of my input waveform, and for this reason, the actual value of the bin locations is nothing more than an indicator of a signal level above the noise floor. Figure 8 shows an FFT output of a simulated dialtone generated and transformed by Mathcad and the output of an actual 'HC11 sampled and transformed dialtone. The 'HC11's result clearly indicates the tones present in the incoming signal.

LISTING 2—continued

```
        BLO     SHIFT1          ; Repeat until done

        SBA                     ; Subtract min value from max value
        LSRA                    ; Divide difference by 2
        ABA                     ; Add the min value to half of the difference
        TAB                     ; Put median into B

        LDX     #DATA           ; Get the address of the data
SHIFT4  LDAA    0,X             ; Get a data point
        SBA                     ; Subtract the offset
        STAA    0,X             ; Save it back
        INX
        CPX     #DATA+255
        BLS     SHIFT4
        RTS
```

LISTING 3
Check.

```
; FFT bin result expectations
; Dialtone = tones 1 and 2
; Ringing = tones 2 and 3
; Busy = tones 3 and 4

TONE1   EQU     71              ; 350 Hz (71-1) * 5
TONE2   EQU     89              ; 440 Hz (89-1) * 5
TONE3   EQU     97              ; 480 Hz (97-1) * 5
TONE4   EQU     126             ; 625 Hz (126-1) * 5
THRESH  EQU     $28             ; Amplitude threshold

* CHECK-This routine looks for the presence of the 4 tones that make up the different
* call progress signals. The amplitude measured will be the sum of the primary bin and
* the one on either side. This will help to include any leakage error in that may have
* resulted from less than exact frequencies or imperfect sampling.

*           A = $00 - no tones present
*           A = $03 - Dialtone    (350 Hz + 440 Hz; bins 71 & 89)
*           A = $06 - Ring signal (440 Hz + 480 Hz; bins 89 & 97)
*           A = $0C - Busy signal (480 Hz + 624 Hz; bins 97 & 126)

CHECK
        LDX     #DATA           ; Get the start of the frequency data
        CLRA                    ; Clear the return variable
        LDAB    TONE1-1,X ; Get bin-1 amplitude for tone 1
        ADDB    TONE1,X         ; Add in the primary bin amplitude
        ADDB    TONE1+1,X ; Add in the bin+1 amplitude
        CMPB    #THRESH         ; Check against threshold
        BLO     CHECK1          ; Branch if not present
        ORAA    #%00000001; Set bit 0 for tone 1 present

CHECK1  LDAB    TONE2-1,X ; Get bin-1 amplitude for tone 2
```

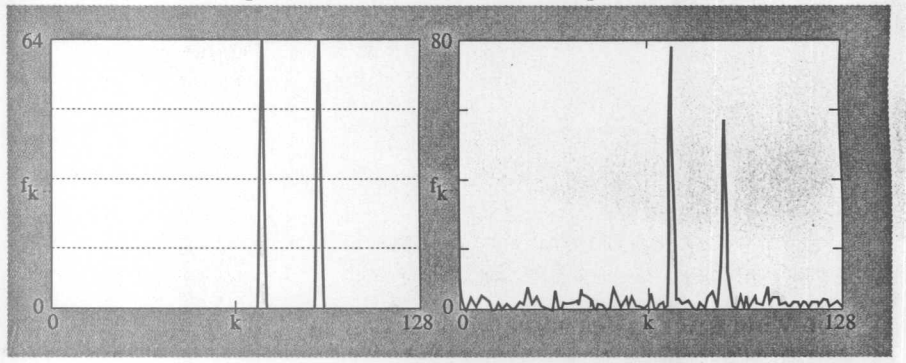
LISTING 3—continued

```
        ADDB    TONE2,X             ; Add in the primary bin amplitude
        ADDB    TONE2+1,X ; Add in the bin+1 amplitude
        CMPB    #THRESH             ; Check against threshold
        BLO     CHECK2              ; Branch if not present
        ORAA    #%00000010; Set bit 1 for tone 2 present

CHECK2  LDAB    TONE3-1,X ; Get bin-1 amplitude for tone 3
        ADDB    TONE3,X             ; Add in the primary bin amplitude
        ADDB    TONE3+1,X ; Add in the bin+1 amplitude
        CMPB    #THRESH             ; Check against threshold
        BLO     CHECK3              ; Branch if not present
        ORAA    #%00000100; Set bit 2 for tone 3 present

CHECK3  LDAB    TONE4-1,X ; Get bin-1 amplitude for tone 4
        ADDB    TONE4,X             ; Add in the primary bin amplitude
        ADDB    TONE4+1,X ; Add in the bin+1 amplitude
        CMPB    #THRESH             ; Check against threshold
        BLO     CHECK4              ; Branch if not present
        ORAA    #%00001000; Set bit 3 for tone 4 present
CHECK4  RTS                         ; Return
```

FIGURE 8
A simulated dialtone spectrum and an actual 'HC11 spectrum.



GIVE IT A TRY

While the FFT has not been thoroughly explained here, I have explored its ease of use and some of its pitfalls. Read more on the concepts of the FFT, but do not be alarmed if you get lost in the integrals. Keep in mind that the FFT is just an efficient algorithm for computing a DFT, and that the DFT transforms a finite series of time domain samples into frequency domain samples. ESP

Clay Dearman is a senior staff engineer with Motorola's Advanced Test Technology group in Austin, TX. He received a BS in electrical engineering from Louisiana Tech University and has been designing embedded systems for about 12 years. He can be reached by telephone at (512) 933-7179, by e-mail at dearman@perch.sps.mot.com, or by regular mail at MS F-15 3501 Ed Bluestein Blvd., Austin, TX 78721.

FURTHER READING

1. Ramirez, Robert W. *The FFT Fundamentals And Concepts*. Tektronix, 1985.

2. Lord, Richard H., "Fast Fourier for the 6800," *Byte*, Feb. 1979, pp 108-119.